# Amusa: middleware for efficient access control management of multi-tenant SaaS applications

Maarten Decat, Jasper Bogaerts, Bert Lagaisse, Wouter Joosen
iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium
first.last@cs.kuleuven.be

## ABSTRACT

Software-as-a-service (SaaS) has been a growing trend in cloud computing for several years. Moreover, SaaS providers are evolving to application-level multi-tenancy, in which all tenants share the application instances, platform and data store with the aim of maximizing resource sharing. For multi-tenant SaaS applications, access control often is the only application-level security mechanism. However, such access control is inherently complex because both the provider and all tenants should be able to specify their access rules for the application. Moreover, these rules must all be securely combined and correctly enforced in the shared multi-tenant application. To address this challenge, we present the Amusa access control middleware. Amusa enables both the provider and all its tenants to efficiently declare their access rules on the SaaS application. To achieve this, Amusa provides incremental three-layered management based on attribute-based tree-structured policies. Afterwards, Amusa securely combines the access rules of all parties and enforces them at run-time with low performance overhead.

## Keywords

Access control, Software as a Service, multi-tenancy, security middleware, performance.

## 1. INTRODUCTION

Over the last years, Software as a Service (SaaS) has drawn increased interest from both industry as well as research communities. SaaS is a type of cloud computing in which tenant organizations rent access to a shared, typically web-based application hosted by a provider [19]. Each of these tenants represents multiple end-users, such as their employees. For the tenants, SaaS promises low management costs. For the provider, SaaS promises lower operational costs by employing *application-level multi-tenancy*.

Application-level multi-tenancy is an architectural strategy in which all tenants share the same code base, application instances and data store [13]. As opposed to the traditional approach of developing and deploying separate versions of the application for each tenant, multi-tenancy aims to reduce the operational costs of cloud applications by maximizing resource sharing.


*SAC'15* April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695708

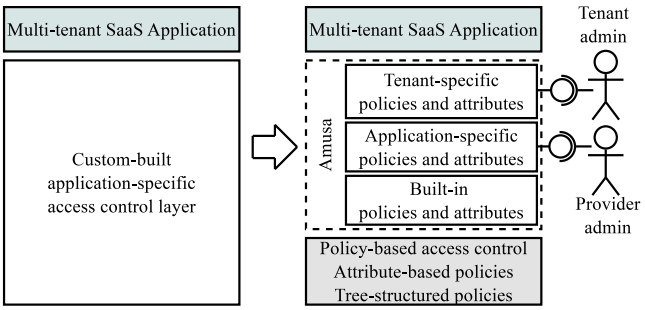


**Figure 1: The Amusa middleware facilitates incremental three-layered management of multi-tenant SaaS applications.**

Access control is the most important and often only application-level security mechanism for SaaS applications. Application-level access control is responsible for constraining the *actions* of authenticated *subjects* on the *resources* in the application by enforcing access rules, for example stating that only sales managers of the European region can send sales offers and only to their assigned customers.

However, access control for multi-tenant SaaS applications is inherently complex: an access control system for multi-tenant SaaS applications should make sure that tenants cannot access each other's resources in the application, should enable the provider to constrain its tenants based on its own rules, should enable the tenants to constrain their end-users based on their own rules, and should enforce the appropriate rules in the shared application at run-time.

Providing this functionality is even more a challenge because the access rules vary from case to case. For example, some applications require tenants to be fully isolated while others require business partners to access each others resources. Similarly, some providers want to constrain tenants based on a pre-paid model, others based on a post-paid model. Additionally, some tenants want to constrain their employees based on their region, others based on contracts, skills or departments. Moreover, if all parties can express their access rules, these rules should be combined securely. For example, tenants should not be able to specify rules that override the provider's policies or give the tenant access to

the resources of other tenants.

As a result of this complexity, the state of practice in SaaS application development does not achieve these requirements. Firstly, the provider's policies about tenants are often hard-coded in the application. Secondly, the tenant-specific access control rules are often supported using a simple role-based scheme. And thirdly, tenant isolation is either hard-coded in the application or relies on isolation at the level of the data-store such as using namespaced silos in Google App Engine [2]. As a result, the provider policies are hard to customize, tenants are able to express only a limited set of access rules, and tenant separation is inflexible.

The state of the art in access control also does not meet these challenges. The combination of policy-based access control [21] with attribute-based [14], tree-structured [7, 17] policies supports the declarative specification of a wide range of rules outside of the application and binding these at run-time. However, these bare technologies still require each SaaS provider to build a multi-tenant access control layer on top of them. And even when employing these technologies, addressing the requirements stated above is not an easy task.

To address these issues, we present Amusa (Access control middleware for Multi-tenant SaaS Applications, illustrated in Figure 1). Amusa allows both the provider and all its tenants to express their access rules for the SaaS application using expressive attribute-based policies. Amusa combines these policies securely and enforces them at run-time. Moreover, Amusa simplifies the overall access control management using an incremental three-layered approach and introduces low performance overhead. Amusa is motivated by two distinct industrial case studies and builds on a large body of work in access control and policy-based middleware.

The remainder of this paper is structured as follows. Section 2 further elaborates on the case studies and summarizes the requirements of Amusa. Section 3 describes Amusa in terms of its employed technologies, its three-layered management architecture and its supporting middleware archictecture. Section 4 evaluates Amusa in terms of security and performance. Section 5 gives an overview of related work. Section 6 concludes this paper.

# 2. MOTIVATION

In this section we motivate and illustrate the challenge of access control for Software-as-a-Service applications based on two case studies of industrial SaaS providers. We first describe these case studies, then illustrate the challenges for access control and conclude with the resulting requirements.

## 2.1 Industrial case studies

This work was performed in collaboration with two industrial SaaS providers in the domains of automated document processing and workforce management, respectively called eDocs and eWorkforce in this paper.

**eDocs.** eDocs offers a service to its tenants to efficiently generate and distribute large numbers of digital personalized documents to their respective users and customers. Typical examples of these tenants are large companies such as banks and press agencies, which distribute pay checks and invoices. Tenants can group and search their submitted documents and can track the receipt of these documents. On the receiving side, users can employ the eDocs platform to read and manage all their received documents.

**eWorkforce.** eWorkforce offers a service to automatically plan the workflows for the product and service appointments of its tenants. Typical examples of these appointments are install and repair jobs for tenants such as large telecom operators, utility companies and retailers. eWorkforce assigns the resulting appointments to the appropriate technicians of its subcontractors, which are the companies executing the actual task. The technicians receive these appointments using the mobile eWorkforce application and afterwards report task progress and consumed resources, such as cables and devices.

Both eDocs and eWorkforce were actively involved in the elicitation of the requirements for the Amusa access control middleware and assisted in its validation. For the interested reader, the detailed description of these two case studies is available in two technical reports [9, 10].

## 2.2 Problem illustration

Access control is the main application-level security mechanism of the applications of eDocs and eWorkforce. Such access control should provide three main functionalities:

1. In the first place, such access control should enable the provider to constrain its tenants, e.g., make sure that only paying tenants can access the application.

2. Moreover, most SaaS applications require that the tenants cannot access each other's resources in the shared application (a form of *tenant isolation* [13]).

3. And finally, both eDocs and eWorkforce have large companies as customers and these companies have stringent security requirements themselves. As a result, the tenants should be able to constrain their own users of the SaaS application.

While this functionality by itself is not trivial, it is made even more challenging by the fact that each party wants to apply its own access rules based on its own specific concepts:

1. In terms of provider rules, the access rules differ between eDocs and eWorkforce: eDocs requires the credit of a tenant to be sufficient to access the application, while eWorkforce charges tenants afterwards.

2. In terms of tenant isolation, both eDocs and eWorkforce require that tenants are separated by default, but also require application-specific and tenant-specific exceptions to this tenant isolation. For example, eWorkforce requires that subcontractors are able to access the tasks assigned to their respective subcontractors, eDocs requires that resellers of the service are able to access the documents of their own customers and some tenants of eDocs require that their business partners can access documents of shared projects.

3. In terms of tenant rules, the access rules differ between tenants. For example, for eDocs, a large bank only permits its users to read documents belonging to their assigned customers and a press agency only permits members of the European region to access the application. For eWorkforce, the tenants constrain their employees based on their skills, interim contracts, projects and internal departments. This variability challenge is related to the known problem of tenant

variability in SaaS [5, 13, 23], but is enlarged because the access rules of a tenant depend heavily on its organizational structure and therefore *inherently* vary from tenant to tenant.

As a result of this complexity, both eDocs and eWorkforce and in extension other SaaS providers are all faced with the challenge of setting up a complex access control infrastructure.

## 2.3 Resulting requirements

The goal of this work is to facilitate building and managing multi-tenant SaaS applications by addressing the challenges illustrated in the previous section. More precisely, the goal of this work is to provide middleware for multi-tenant access control management. This middleware should provide efficient access control management to the providers of SaaS applications and their tenants, and should be reusable by multiple SaaS providers. As such, the requirements for this middleware are five-fold:

1. the middleware should enable the provider to easily constrain the tenants in terms of application-specific concepts,
2. the middleware should enable each tenant to easily constrain its users in terms of its tenant-specific concepts,
3. the middleware should combine the policies of all involved organizations securely,
4. the middleware should enforce the access rules appropriate for each request in the shared application and,
5. the middleware should introduce low performance overhead for the SaaS application.

# 3. AMUSA: SECURE THREE-LAYERED ACCESS CONTROL MANAGEMENT

To address the requirements stated in the previous section, this paper presents the Amusa middleware. Amusa enables both the provider and all its tenants to express their access rules on the SaaS application, combines these rules securely and enforces them at run-time. To allow all parties to express their rules in terms of their own concepts, Amusa leverages the expressive model of attribute-based access control. Moreover, in order to simplify the overall access control management of all parties involved, Amusa employs an incremental three-layered approach in which the provider builds on the attributes and policies defined by Amusa, and the tenants build on the attributes and policies defined by the provider. In this section, we describe the enabling technologies leveraged by Amusa, its three-layered access control management, how Amusa securely combines all policies and its supporting middleware architecture.

**Key scenario.** For the rest of this paper, we focus on an illustrative scenario of the eDocs case study employing two tenants: Large Bank and Press Agency. In this scenario, eDocs requires the credit of a tenant to be sufficient to access the application and relaxes the default tenant isolation policy to permit resellers of the eDocs application to view the documents of their respective tenants. Large Bank only permits its users to read documents belonging to their assigned customers and Press Agency only permits members of the European region to access the application. Moreover, Large Bank relaxes the tenant isolation policy to permit its business partners to access the documents of shared projects.

## 3.1 Enabling technologies

To achieve the requirements of the previous section, Amusa builds on three state-of-the-art access control technologies that support part of these requirements, i.e. policy-based access control, attribute-based access control and tree-structured policies.

**Policy-based access control.** Policy-based access control is an approach in which the access control rules are separated from the mechanisms that enforce them. As such, they can be externalized from the application that they constrain and be expressed in modular, declarative *access control policies* [21]. Amusa employs policy-based access control to enable the tenant and provider to specify their own rules without having to change the application.

**Attribute-based access control.** Attribute-Based Access Control (ABAC [14]) is a recent model to express access rules in terms of key-value properties of the subject, the resource, the action and the environment. These properties are called *attributes* and include for example the subject identifier, subject roles, resource type and the time. Amusa employs ABAC because attributes provide a simple abstraction that enables users to be managed in terms of their properties. Moreover, ABAC provides an expressive policy model that is able to express most of the rules of our case studies such as permissions, roles, ownership, time, separation of duty and location. However, ABAC by itself does not provide efficient access control management, e.g., each attribute for each subject or resource still has to be defined by the appropriate party.

**Tree-structured policies.** Tree-structured policies or *policy trees* are a means to structure multiple rules into one well-defined policy and reason about possible conflicts between these rules (e.g., [7, 17]). To achieve this, every element in the tree defines to which requests it applies by means of a *target*. The rules are the leaves of the tree and decisions of children are combined using combination algorithms such as *FirstApplicable* and *PermitOverrides*. Amusa employs policy trees to combine the policies of the tenants and the provider while guaranteeing important security properties, e.g., making sure that tenants cannot override the provider policies.

As a result, Amusa is an access control middleware that builds on these technologies, but adds a SaaS-specific layer that enables flexible and secure multi-tenancy. We discuss the three-layered access control management of Amusa in the next section.

## 3.2 Three-layered access control management

In terms of attribute-based access control, access control management consists of managing attributes and managing policies that employ these attributes. Amusa divides this management over the three kinds of stakeholders involved: the provider, the tenants and the Amusa middleware itself. Firstly, Amusa predefines common attributes and policies that can be reused across applications, providers and tenants. Secondly, the provider offers the SaaS application.

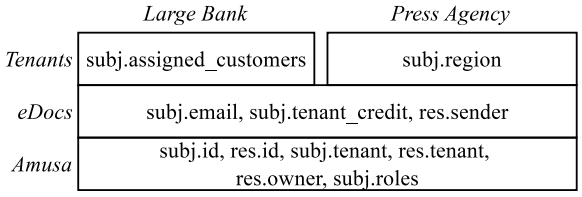


**Figure 2: Amusa enables attributes to be incrementally defined in three layers: Amusa, the provider and the tenants (illustrated for the key scenario).**

Therefore he knows the application domain and manages the application resources and actions that should be protected. This results in application-specific attributes and policies. The provider also offers the service to tenants and therefore also manages the policies to constrain them. Thirdly, the tenants manage their own users based on organization-specific attributes and policies. Each of the latter parties builds on the previous layer. This gradual extension favors reusability and simplifies the overall management effort. In the rest of this section, we describe the three-layered attribute management and policy management in more detail.

### 3.2.1 Three-layered attribute management

Managing attributes entails two kinds of actions: (1) defining possible attributes for subjects and resources and (2) assigning values to the attributes. In this case, the three-layered management applies to attribute definition: Amusa itself pre-defines a fixed set of attributes, which the provider can extend for its own application and which the tenants in turn can extend for their organization. More precisely, Amusa pre-defines the attributes it requires for its correct functioning and a number of frequently occurring attributes. The former includes the subject and resource identifiers and the associated tenant of a subject or resource, the latter includes the owner of a resource and the roles of a subject. The provider then defines the attributes of the resources in its application and optionally some frequently-used application-specific subject attributes that can be used by all tenants. For example, in eDocs, each document has a sender and a destination, each user has an e-mail address and each tenant has a credit. Finally, the tenants define their tenant-specific subject attributes. For example, Large Bank defines attributes for departments, teams and projects, and Press Agency also for geographic regions. The resulting attribute definitions for the key scenario are illustrated in Figure 2.

After defining the appropriate attributes, each stakeholder is responsible for assigning them to the resources and subjects it controls. Amusa automatically assigns attributes where possible, e.g., the identifier and tenant of every new subject it creates. The provider (or more specifically, the SaaS application) then assigns the appropriate attributes to its resources. In essence, these attributes are already present in the SaaS application itself. Finally, the tenant assigns the subject attributes defined by Amusa, the provider or itself to its subjects.

### 3.2.2 Three-layered policy management

The same three-layered model described above also applies to policy management. The policies of each layer can be specified in terms of attributes that are available in that

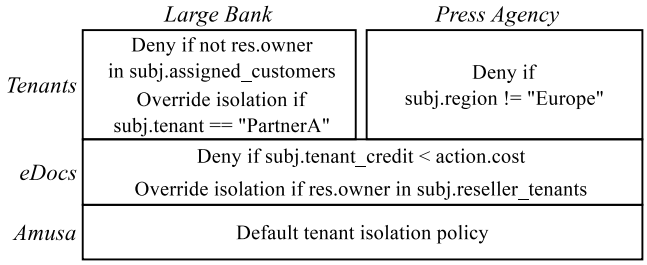


**Figure 3: Next to attributes, Amusa also enables policies to be incrementally defined in three layers: Amusa, the provider and the tenants (illustrated for the key scenario).**

layer. Firstly, Amusa itself has some policies built-in that apply to the provider and all tenants. The most important of these is the default tenant isolation policy. These policies can only reason about the attributes pre-defined by Amusa. Secondly, the provider can specify policies about the tenants as a whole. These policies can employ the attributes specified by Amusa and the provider itself. For example, eDocs specifies that users belonging to a certain tenant cannot send any document if the credit of that tenant is not sufficient. Thirdly, the tenants can specify policies that apply to their own users. These policies can employ the attributes specified by Amusa, the provider and the tenant itself. For example, Press Agency only permits subjects of the European region to access the application. In addition to these policies, the provider and the tenants can also specify selective exceptions to tenant isolation. For example, this enables eDocs to permit resellers to access the documents of their customers and Large Bank to permit its business partners to access documents of shared projects. The resulting policy definitions for the key scenario are illustrated in Figure 3. The policies for constraining tenants and users are combined using logical "and" so that the provider and tenants can incrementally restrict access. The isolation exceptions on the other hand are combined using logical "or" so that the provider and the tenants can incrementally expand access.

## 3.3 Securely combining the policies

Amusa is responsible for combining the policies of all stakeholders in such a way that they are enforced correctly. This means that even though all stakeholders can customize Amusa by defining their own policies, Amusa must still guarantee certain security properties. Most importantly, tenants should not be able to leverage there own policies to override tenant isolation or the rules of the provider.

In order to achieve these properties, Amusa combines the rules of all parties using the policy tree shown in Figure 4. The leafs of this tree represent rules that return *Permit* or *Deny* on a certain condition. The intermediate nodes combine the effects of their children using a combination algorithm and specify to which requests they apply using a target. When the provider or a tenant adds or modifies a policy, Amusa constructs this policy tree as follows:

1. Build the sub-tree for tenant isolation:
   - Create a policy with target "*any*" and combination algorithm *PermitOverrides*.
   - Add the default rule for strict tenant isolation.
   - Add the provider exceptions to tenant isolation.

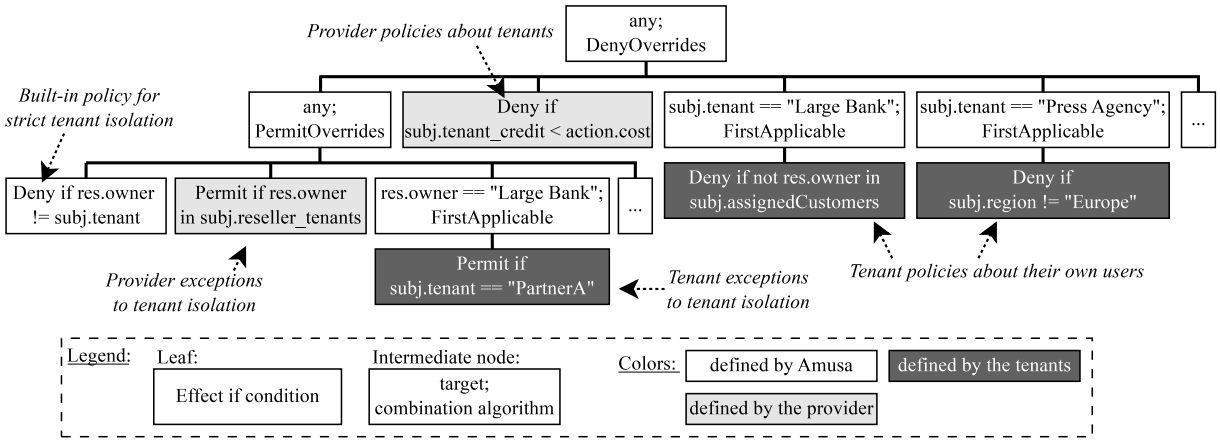


**Figure 4: The policy tree that securely combines the policies of all stakeholders, illustrated for the key scenario.**

- For each tenant, add its isolation exceptions wrapped in a policy with target "*res.owner == <tenant_id>*".

2. Build the complete policy-tree:
   - Create a policy with target "*any*" and combination algorithm *DenyOverrides*.
   - Add the sub-tree for tenant isolation.
   - Add the provider policies about tenants.
   - For each tenant, add its policies about its own users, wrapped in a policy with target "*subj.tenant == <tenant_id>*" and the combination algorithm chosen by the tenant.

This policy tree ensures that the overall access control decision is correct because of the employed policy combination algorithms. Firstly, the sub-tree for tenant isolation, the policies of the provider about tenants and the policies of each tenant about their users are combined using *DenyOverrides*. As such, a request is only permitted if both tenant isolation, the provider and the tenant permit it. On the other hand, the default tenant isolation rules and the exceptions of the provider and the tenants are combined using *PermitOverrides*. As such, a request is permitted if one exception permits it (and if the other top-level policies permit it). Additionally, the policies of each tenant are inserted in the tree below a target that only applies to the subjects or resources of that tenant and cannot be modified by the tenants. As such, only the policies of the appropriate tenant apply to the employees of that tenant. The security evaluation of Section 4.1 validates and illustrates the security properties that follow from this policy tree.

Notice that every element in the policy tree of Figure 4 is defined by exactly one stakeholder. This illustrates that three-layered access control management can effectively segregate the different stakeholders.

## 3.4 Supporting middleware architecture

Following the management architecture described in the previous sections, this section describes the architecture of the Amusa middleware that supports this management. Figure 5 shows this architecture. From the point of view of Amusa, the application consists of two components: the application logic and the database containing the application resources. The Amusa middleware itself consists of six major components: the policy decision point, the authentication

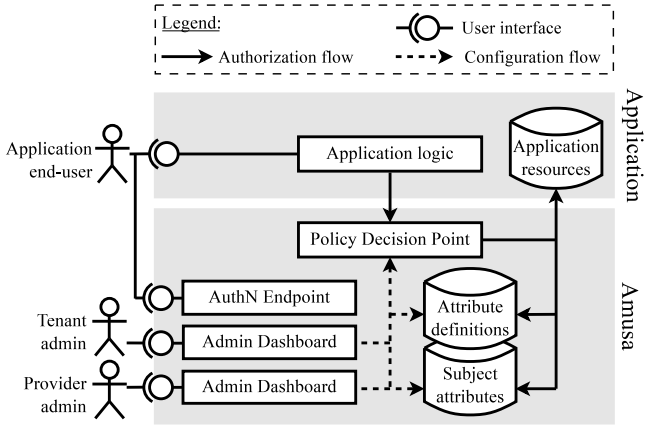


**Figure 5: Architecture of the Amusa middleware. The policy decision point is the component that evaluates the policies and returns an access decision. AuthN is authentication.**

endpoint, the provider administrator dashboard, the tenant administrator dashboard, the database containing attribute definitions and the database containing subject attributes. Of these components, the policy decision point evaluates the access control policies and returns access decisions to the application. The dashboards are used by the provider and tenant administrators to define and assign attributes, and deploy policies. Whenever an administrator makes a change to its policies, the dashboards construct the complete policy tree and deploy it to the policy decision point.

The authentication and authorization flows resulting from this architecture are as follows. In order to authenticate a user, the application redirects the user to the authentication front-end provided by Amusa using a federated authentication technique such as SAML [1]. After successful authentication, the authentication endpoint redirects the user back to the application with an authentication statement and his or her subject attributes. In order to then authorize a request of the user, the application sends an authorization request consisting of attributes to the policy decision point. The policy decision point then evaluates the policy tree of Figure 4. When the policy decision point requires

an attribute during policy evaluation, it first checks the attributes sent by the application and then searches for it in the appropriate database if needed. In the end, the policy decision point returns an access decision to the application, which enforces this decision.

Note that in essence, the policy decision point evaluates the complete policy containing all policies of all stakeholders for every request. However, because of the structure of the complete policy tree (see Figure 4), only the appropriate policies apply and are effectively evaluated. As such, this set-up effectively binds the correct policies at run-time, which was one of the explicit requirements for supporting multi-tenant access control.

# 4. EVALUATION

The previous section presented the concept of three-layered access control management for SaaS applications and the supporting Amusa middleware. Section 3.2 already illustrated that this three-layered approach can effectively segregate the different roles in SaaS access control management. This section further evaluates the security properties and performance impact of Amusa.

## 4.1 Security

Section 3.3 described the policy tree employed by Amusa to securely combine the policies of all the involved parties. In this section, we validate that this policy tree indeed keeps the complete policy secure. First we deduct a number of security properties guaranteed by the policy tree, then we illustrate how these properties mitigate certain misuse cases.

**Security properties.** In essence, the policy tree guarantees the following security properties:

*1. If the provider denies a request, a tenant can never override this.* The policy tree achieves this by combining the policies of all stakeholders using the *DenyOverrides* combination algorithm. As a result, a *Deny* of the provider policies can never by overridden by a tenant policy.

*2. Tenants can only specify policies about their own users.* The policy tree achieves this by inserting the policies of a certain tenant below a target that only applies to its own *subjects*.

*3. Only the policies of the appropriate tenant are taken into account for a certain request.* The policy tree achieves this as a result of the previous guarantee, combined with the guarantees of Amusa that all tenant identifiers are unique, that the tenant is correctly assigned to subjects and that the assigned tenant cannot be changed by any subject.

*4. Tenants and provider can override the default tenant isolation policy.* The policy tree achieves this by combining the default tenant isolation policy with the isolation exceptions of the provider and the tenants using *PermitOverrides*. As such, the provider and its tenants can all override a *Deny* of the default isolation policy.

*5. Tenants can override tenant isolation only to permit others to view* its *application resources.* The policy tree achieves this by inserting the isolation exceptions of a certain tenant below a target that only applies to its own *resources*.

*6. Tenants cannot gain access to the resources of other tenants using their own policies.* The policy tree achieves this as a result of the previous security guarantees. More precisely, if a user of a tenant A tries to access a resource belonging to tenant B, the constraining policies of tenant A and the isolation exceptions of tenant B apply, respectively because of the subject and the resource of the request. However, unless the isolation exceptions of tenant B permit the request, the *Deny* of the default isolation policy will always overrule a possible *Permit* of tenant A.

Note that some of these guarantees also depend on the correctness of certain attributes. For example, it should not be possible for a user to change the attribute *subj.tenant_credit*. To guarantee this to the provider, the supporting middleware can enforce that certain attributes cannot be defined or assigned by tenants.

**Illustration.** To show that these properties keep the complete policy secure, take the following three examples:

*Example 1.* Assume that Large Bank tries to gain access to the resources of Press Agency by configuring the following rule *R1: Permit if subj.tenant == "Large Bank" and res.owner == "Press Agency"*. This is handled by Guarantee 6, keeping the resources of Press Agency secure.

*Example 2.* Imagine that Large Bank tries to perform more actions than its credit permits by configuring the following rule *R2: Permit if subj.tenant_credit == 0*. This is handled by Guarantee 1, which gives preference to the *Deny* of the provider policies.

*Example 3.* Imagine that Large Bank tries to deny the use of the application to Press Agency by configuring the following rule *R3: Deny if subj.tenant == "Press Agency"*. This is handled by Guarantee 2, i.e. *R3* will never apply to the users of Press Agency.

## 4.2 Performance

Next to the security properties of Amusa, we also evaluate its performance overhead on a request of a user to the application. More precisely, we evaluated the overall performance overhead of Amusa and the behavior of Amusa with regard to a growing number of tenants.

**Set-up.** To evaluate the performance overhead of Amusa, we developed a prototype of both the Amusa middleware as well as the eDocs application running on top of this middleware. Both prototypes are written in Java and employ the Spring 3 Web MVC framework for the front-ends. The Amusa prototype employs SAML [1] for authentication, XACML2 [20] for policy specification and an extended version of the SunXACML engine for policy evaluation. The application prototype allows users to send documents to each other, as well as reading and managing these documents.

The tests deploy the architecture of Figure 5 on three nodes, respectively hosting (1) the application logic and database, (2) the Amusa attribute database, and (3) the client making the requests to the application. The policy decision point is compiled into the application. Each test was repeated until the confidence interval of the average policy evaluation time was situated within 2% of the sampled mean for a confidence level of 95%. We excluded the top 1% of the results because a small fraction of these were up to 100 times larger than the mean, presumably because of running the tests on a shared cloud platform.

The tests employ the policies of the eDocs SaaS application and its Large Bank tenant. Measuring the performance

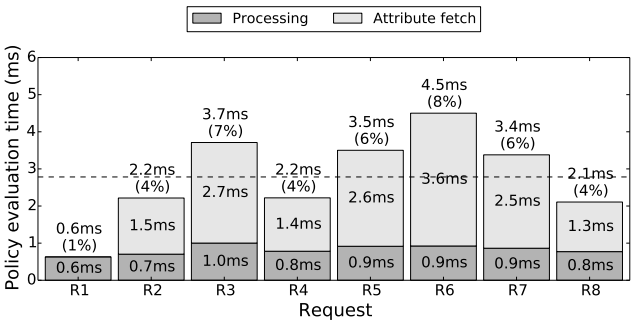


**Figure 6: The total policy evaluation time from the point of view of the application for 8 representative requests that cover the whole policy, showing the time spent on fetching attributes and on processing the policy. The percentage on top of each bar represents the fraction of the complete time to process the application request spent on policy evaluation. The dotted line represents the average over all requests. Lower is better.**

overhead of access control is not trivial because this overhead largely depends on the size and structure of the involved policy. In this regard, we opted for measuring the performance of a set of realistic policies instead of a set of artificial policies. The employed policy contains 32 rules, has a tree depth of 4, requires 26 different attributes and comprises 1119 lines of XACML in total. Because reaching an access decision often does not require to evaluate the complete policy, we report the results for 8 representative authorization requests that cover the complete policy. More precisely, these requests trigger the tenant isolation policy, the provider policies and the different rules of Large Bank.

A demo of Amusa, the code of both prototypes and the employed policies are available on-line[1].

**Overall performance overhead.** To measure the performance overhead of Amusa, we measured the time it takes to evaluate the policy for every request. Figure 6 summarizes these results. As shown, the overall performance overhead of Amusa is low with an average of 2.8ms for all requests. In a broader perspective, this is 4.9% of the server-side application request time and 1.5% of the total client-side request time. Figure 6 also shows that the policy evaluation time varies from request to request and is largely determined by the time spent on fetching attributes. An attribute fetch requires 1.2ms on average and the total time for fetching attributes is determined by the number of attributes required for reaching a decision. R6 requires the most attributes (i.e., 7) and as a result requires the longest evaluation time.

**Growing number of tenants.** With regard to a growing number of tenants, the only aspect of Amusa that grows with the number of tenants is the size of the complete policy tree. More precisely, each new tenant adds a branch to the policy tree of which the applicability will be checked during each policy evaluation. eDocs and eWorkforce both have around 50 tenants. For this size, testing the applicability of all tenant policies imposes a mean performance overhead of less than 0.15ms. This overhead is negligible as compared to the overall performance overhead, but grows linearly with the number of tenants. As such, it can become beneficial for larger SaaS applications to introduce a specialized policy primitive for more efficient matching of the tenant policies, e.g., a hash-map based on the tenant identifier.

**Summary.** These tests can be considered worst-case given that the operations of our application were limited in complexity with respect to practical applications. Even in this context, our performance results can be considered low. Moreover, the overhead of Amusa can still be lowered using other performance tactics, e.g., high-performant policy engines such as [18], which was not the focus of this work.

# 5. RELATED WORK

The Amusa middleware builds on a large body of work from the domains of multi-tenancy, access control and policy-based middleware. Firstly, while SaaS is a relatively young paradigm, it has been subject of research for quite some time. For example, in 2007, Guo et al. [13] identified two high-level requirements: isolating tenants and allowing the SaaS application to be customized to the specific needs of each tenant. The latter is also identified by Bezemer et al. [5] and Sun et al. [23]. This work focuses on access control, which is both a means to provide (application-specific) tenant isolation and an important source of variability in SaaS. However, to the best of our knowledge, very little work has been performed to achieve these requirements. In the state of practice, the authors are not aware of solutions that provide expressive tenant-specific policy-based access control, while our industrial partners stressed the need for such technology. Tenant isolation is mostly implemented manually, and builds on strict data isolation in the data store. More specifically, a built-in tenant identifier is used in database queries. This approach is adopted for example in GAE [2]. However, this does not allow easy application-specific customization of the isolation policy. In the state of the art, Calero et al. [3] also focused on multi-tenant authorization for cloud applications. They opted for extending role-based access control (RBAC, [12]) specifically for multi-tenancy. This work has later been formalized by Tang et al. [24]. Amusa extends this approach by moving from RBAC to more expressive attribute-based policy trees and extending the architecture into a reusable middleware.

Secondly, Amusa was inspired by other access control systems described in literature. Multiple such systems have been described in the domain of grid computing, e.g., CAS, Cardea and PRIMA [6]. Access control in this domain focuses on scalable access control management for a possibly large number of possibly large *virtual organizations*. Therefore, these systems employ techniques similar to the ones used by Amusa, such as the decoupling of enforcement and policy evaluation. A good overview of this domain is given in [6]. Amusa combines these techniques with the recent technologies of ABAC and policy trees into a configurable access control middleware for the domain of SaaS. More recently, Fatema et al. [11] and Lazouski et al. [16] also described access control systems relevant to Amusa. Both complement Amusa because they employ similar building blocks, but have a different focus, respectively privacy in multi-organizational systems and usage control in Infrastructure as a Service. Therefore, it would be interesting to see how these systems can be combined with Amusa.

Thirdly, Amusa builds on the experience with policies in

[1]https://distrinet.cs.kuleuven.be/software/amusa/

the domain of middleware. For example, early work by Sloman [22] already applied policies for declaratively managing access control in distributed systems, which later lead to the definition of the influential Ponder specification language for access control policies [8]. Next to access control, policies have been applied for a large variety of goals in the domain of middleware. Amongst others, Bacon et al. [4] employ policies for information flow control in multi-domain applications, Wun and Jacobson [25] for managing content-based publish/subscribe middleware and Kumar et al. [15] for describing self-management behavior. The common denominator of all this work is that policies are used to separate semantics from enforcement and describe the semantics declaratively. Amusa applied this principle to the domain of SaaS access control.

# 6. CONCLUSION

In this paper, we presented Amusa, an access control middleware for multi-tenant Software-as-a-Service applications. This research was conducted in close collaboration with two industry partners, an approach that resulted in a set of key requirements for access control in SaaS. Amusa offers a management and enforcement architecture that supports these requirements. Both the provider and the tenants can effectively specify their access rules in terms of their own concepts using three-layered access control management based on attribute-based tree-structured policies. Amusa combines the rules of all parties securely and enforces them at run-time. Moreover, the evaluation showed that Amusa also introduces low performance overhead.

The main contribution of Amusa is the SaaS-specific layer for flexible and secure multi-tenancy on top of attribute-based tree-structured access control policies. This work applied these techniques in a three-layered approach, which simplifies access control management through gradual refinement. To the best of our knowledge, this work is the most extensive study and application of these technologies described in literature and our experience leads us to believe that all three technologies are important enablers for future access control research.

**Acknowledgments**. This research is partially funded by the Research Fund KU Leuven, by the EU FP7 project NESSoS and by the Agency for Innovation by Science and Technology in Flanders (IWT). With the financial support from the Prevention of and Fight against Crime Programme of the European Union (B-CCENTRE).

# 7. REFERENCES

[1] Security Assertion Markup Language (SAML) v2.0. `http://www.oasis-open.org/standards#samlv2.0`, March 2005.

[2] Namespaces Java API - Google App Engine - Google Developers. `https://developers.google.com/appengine/docs/java/multitenancy/`, May 2014.

[3] J.M. Alcaraz Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. Toward a multi-tenancy authorization system for cloud services. *Security Privacy, IEEE*, 2010.

[4] J. Bacon, D. Evans, D. Eyers, M. Migliavacca, P. Pietzuch, and B. Shand. Enforcing end-to-end application security in the cloud. In *Middleware*. 2010.

[5] C.-P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. 't Hart. Enabling multi-tenancy: An industrial experience report. In *ICSM*, 2010.

[6] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori. Access and usage control in grid systems. In *Handbook of Information and Communication Security*. 2010.

[7] J. Crampton and M. Huth. An authorization framework resilient to policy evaluation failures. *ESORICS*, 2010.

[8] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder policy specification language. *IEEE POLICY*, 2001.

[9] M. Decat, J. Bogaerts, B. Lagaisse, and W. Joosen. The e-document case study: functional analysis and access control requirements. Technical report, KU Leuven, 2014.

[10] M. Decat, J. Bogaerts, B. Lagaisse, and W. Joosen. The workforce management case study: functional analysis and access control requirements. Technical report, KU Leuven, 2014.

[11] K. Fatema, D. Chadwick, and S. Lievens. A multi-privacy policy enforcement system. IFIP. 2011.

[12] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *TISSEC*, 2001.

[13] C.J. Guo, W. Sun, Y. Huang, Z.H. Wang, and B. Gao. A framework for native multi-tenancy application development and management. In *CEC/EEE*, 2007.

[14] V. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. *NIST Special Publication*, 2014.

[15] V. Kumar, B.F. Cooper, G. Eisenhauer, and K. Schwan. imanage: Policy-driven self-management for enterprise-scale systems. Middleware, 2007.

[16] A. Lazouski, G. Mancini, F. Martinelli, and P. Mori. Usage control in cloud systems. In *Internet Technology And Secured Transactions*, pages 202–207, 2012.

[17] N. Li, Q. Wang, W. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin. Access control policy combining: Theory meets practice. In *ACM SACMAT*, 2009.

[18] A. Liu, F. Chen, J. Hwang, and T. Xie. XEngine: a fast and scalable XACML policy evaluation engine. In *ACM SIGMETRICS*, 2008.

[19] P. Mell and T. Grance. The NIST definition of cloud computing. *NIST*, 2009.

[20] T. Moses et al. eXtensible Access Control Markup Language (XACML) 2.0. *OASIS Standard*, 2005.

[21] P. Samarati and S. de Vimercati. Access control: Policies, models, and mechanisms. In *Foundations of Security Analysis and Design*. 2001.

[22] M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 1994.

[23] W. Sun, X. Zhang, C.J. Guo, P. Sun, and H. Su. Software as a service: Configuration and customization perspectives. In *IEEE SERVICES*, 2008.

[24] Bo Tang, R. Sandhu, and Qi Li. Multi-tenancy authorization models for collaborative cloud services. In *CTS*, pages 132–138, May 2013.

[25] A. Wun and H.-A. Jacobsen. A policy management framework for content-based publish/subscribe middleware. In *Middleware*. 2007.